Jails by Java:

Simulating Racial Disparities in Incarceration Rates and Subsequent Solutions with Object-Oriented Statistical Programming

December 14, 2015

Simulating Racial Disparities in Incarceration Rates and Subsequent Solutions with Object-Oriented Statistical Programming

Munawar S. Rahman, 84-70 129th St. Apt#1S Kew Gardens, NY 11415

Stuyvesant High School, New York City, NY

Teacher: Ms. Ellen Schweitzer

ABSTRACT

Using Java, I built an agent—based model of incarceration with data collected from the 2014 Lum et al. study, utilizing principles of object-oriented programming to construct a synthetic population and statistical functions provided by CERN's Jet library to generate data which illustrates the racial disparity in incarceration rates. The goals of the development of this model are threefold: First, to demonstrate the viability of porting and optimizing a model for another programming language, which has the potential of increasing accessibility, expandability and comprehension of the model. Second, leading in from the first one, is the intent of reproducing the observed disparities in incarceration rates through minor parameter adjustment, providing a demonstration of the contagious nature of the model. The third goal is an expansion of the Lum et al. model by simulating potential solutions of equalizing sentencing between white and black populations as a means off alleviating the immense disparity, the relative simplicity of the model allows one to generate such ideal conditions, although the parameters can be further adjusted to account for more realistic solutions. The ultimate end of this work is to reinforce the value of computational agent-based models in epidemiology and criminology.

TABLE OF CONTENTS

Introduction

Procedural simplification allows researchers to parse the noise of reality; complex phenomena and intricate human interaction can be boiled down to sets of agents and variables to be generated and observed. As computing power grows exponentially, so does the intrinsic value of computational modelling in simulating human systems. The better we are able to synthesize the interactions between agents in a population, the better we are capable of elucidating the relationships between variables and formulating concrete solutions to social dilemmas. Agent-based models in particular are one class of computational models which simulate the interactions

of autonomous agents, either individual or collective entities, with the purpose of determining the effects of their behavior on large-scale consequences, provided each agent acts similarly.

In the 1996 Epstein and Axtell study [6] the two use a bottom up approach with a computer simulation titled "Sugarscape", where they model the behavior of synthetic population with preassigned attributes, and have observed that collective behaviors emerge as a result of the minute interactions of agents within the model. As demonstrated in the Hommes study [7] featuring economic simulations it is possible to use such bottom up approaches, where agents follow simple rules, to model stylized facts. A stylized fact is an economic term to refer to consistent qualitative empirical findings often acknowledged as truth e.g. a positive relationship between education and income, and these stylized facts are often emergent properties of simple agent-based models. According to the Windrum study [10], agent-based models contain three ingredients. First, a bottom up approach to producing aggregate properties, as elucidated in Epstein and Axtell. Second, the presence of "boundedly-rational agents" which follow specific sets of rules, allowing for relative parsimony and ease of observation. Third is networked direct interactions, describing how agents are not merely static within their populations but must interface with each other to generate the data which demonstrate emergent properties.

Computational agent-based modelling has seen further usage in the field of criminology as a supplement to traditional methods, as demonstrated in the 2012 Birks et al. study [4] which tested hypothetical mechanisms which could offer mechanisms for the "stylized facts" of criminal phenomena. In the development of this simulation, Birks et al. create a simulation of residential burglary with a synthetic population of potential targets and offenders (boundedly-rational agents) who behave according to theories of environment criminology (networked direct interactions) from which a series of experiments is run analyzing the impact on these interactions on patterns of offending (bottom-up approach). The methodology of this model is similar to that of and most likely provided the basis for the 2014 Lum et al. study [8] which involved a similar procedure towards the goal of explaining racial disparities in incarceration rates. The research of Lum et al. is one of many in the past decade of criticism and analysis of the prison-industrial complex, specifically concerning the racist nature of mass incarceration and its consequences in perpetuating a cycle of poor material, social and psychological conditions [2].

The notion that discrepancies in sentencing can produce huge disparities in rates of imprisonment has virtually become a stylized fact within the field of criminology, though it is

difficult to grasp due to the seemingly irrational nature of inequitable sentencing in the first place. It should be understood, however, that inequitable sentencing is the product of years of differences in the material conditions between blacks and whites. Even in the early years of the War on Drugs there were already more blacks in prison, this was further exacerbated by disproportionate policing. According to data provided by the Bureau of Justice Statistics Statistical Analysis Tool, the per capita rate of incarceration has grown from 137 to 511 per 100,000 persons since the 1970s. By 2011, Black incarceration rates were over six times higher than White rates- 3023 per 100, 000 for Blacks, and 478 per 100, 000 for Whites. The increase in racial disparity for Blacks has not coincided with any supposed growth in overall Black criminal activity [9], such disproportionality can also be explained as either active discrimination- such as the use of incarceration to deal with a "racial threat"- or passive, such as increased police surveillance on low-income communities with certain demographics. A meta-analysis shows that sentencing inequalities also contribute to this, even when accounting for social contexts and legal factors, Blacks receive longer sentences than Whites even when committing the same crime e.g. drug offences [3]. The nomenclature of incarceration "epidemic" [2][8] is justified not just by the growth of the carceral state, but also by the extent to which the probability of imprisonment is "transmitted" amongst friends and family of an individual who has been incarcerated or released [5].

After taking all of this into account, along with aligning with the three-tiered approach to constructing agent-based models, Lum et al. have successfully observed marked inequalities in rates of imprisonment by race through minute adjustment of initial parameters, including sentence length and the initial incarcerated population for black and white communities. Thus, the purpose of this work is to extend the usage of their agent-based model through reverse-engineering the code used for the simulation towards a more fully object-oriented version of their model, which is more consistent with the notion of agent-based modelling, with the aim of reproducing the stylized facts they have observed and generating potential solutions to the dilemma of mass incarceration. This new implementation of the model will invoke Java, a general-purpose programming language known for being very object oriented. While Python, the language the original model is in, can also be utilized as an object-oriented programming language, the implementation utilized in Lum et al. is not very intuitively object based, at the very least if this work is able to reproduce the disparities in incarceration rates and toy with plausible alleviations to the epidemic then it is worth creating more models in as many programming languages as possible.

IMPLEMENTATION

A proper simulation [11] has a bare minimum of two components- the synthetic population and the set of functions which will be run on the former in order to generate data. Within the context of object-oriented programming, an object would represent a single individual, while a data structure composed of these objects would be the entire population. There are numerous advantages in representing individual members of the population as mutable objects: first, it is intuitive and thus implementable amongst a variety of object-oriented languages. Second, each object can possess personal attributes represented as instance variables, allowing for easier manipulation of the characteristics of each individual. Finally, this makes it easier to iterate over individual members of the population to adjust their attributes based on already existing ones, for example, to implement a conditional statement where one variable is modified based on the value of another.

Attributes for a synthetic population are collected from the Lum et al study, which generated a synthetic network of agents represented in the form of a nested Pythonic dictionary. The Java equivalent of Python dictionaries are Hashmaps which perform a similar function of mapping keys to values. However, the Java implementation of Hashmaps is too convoluted and syntax heavy to construct a reliable, usable synthetic population.

This is where Java classes come in handy, since they can be initialized as objects with their respective attributes. A class titled Individual is created which implements the Serializable interface- this means that objects of this class can be serialized i.e. compressed into a stream of bytes so it can be utilized by a different Java file while saving memory. The instance variables of the class coincide with the attributes assigned to individual members of the population. These include integer values of the ID number of the individual, the iteration number when the person is set to die, the iteration number for when the person is set to be born, the iteration number the person meets their partner the iteration number of when the individual is infected, the age at which the individual has their first child, and the length of the individual's sentences. In addition, there is a String value reporting the sex of the individual- either male or female.

Packaged within each class of Individual are also instance variables which report the relationship of one agent to another via the ID numbers of the respective agents. For instance, the integer variable "partner" reports the ID number of the agent's partner, while Integer arrays titled "children", "parents", "siblings" and "friends" contain the ID numbers of the other individuals in

the population who hold that specific relationship to the individual. The code implementation is shown below in Figure 1.1.

```
import java.io.*;

public class Individual implements Serializable {
    //instance variables
    String sex;
    int id;
    int death;
    int birth;
    int partner;
    int iter_married;
    int infected;
    int age_at_first_birth;
    int incarcerated = 0;
    int[] children;
    int[] parents;
    int[] siblings;
    int[] friends;
    double[] location;

    public Individual() {}
}
```

Figure 1.1

The only feat accomplished, however, is the construction of just one individual in a population, not the latter itself. However, just as arrays of integers have been created which generate a list of integers, Java allows one to initialize arrays of objects which represent lists of that particular object. This requires creating a separate Java class- titled PopuCreator- and initializing a new instance variable *popu* , representing a new array of the class of Individuals with a population count of 8856. Since each element in the array contains all the attributes of the Individual class, iterating over each element in array permits the modification of each attribute of every single Individual.

A separate Python program deserializes the original synthetic population created in the Lum et al. study, extracts the values of every single attribute from every single Individual in the population, and imports them into a series of comma-separated-value files containing all the data. Within the main method of the PopuCreator class, I write a series of functions which reads the data from the individual text files and inserts them into a series of arrays for the appropriate attribute. This is done by initializing a BufferedReader to read a text file, e.g. “birth.txt” (a list of integers separated by commas) and then inserting each value one-by-one into an array with a value of 8856, with one attribute per individual. With all the data having been collected, the final step in creating a usable population is to iterate over the currently empty array of objects of Individual class and assign a value to each individual’s attributes. Now that there is a gigantic population filled

with 8856 Individuals with the appropriate characteristics, the default Java serialization methods are used to compress it into a stream of bytes so that it may be read by a different file.

After these steps, it is now time to write a new Java class containing all the methods which will operate on the newly constructed population. This class will import a few Java libraries as well as CERN's Jet and Colt libraries which provide statistical methods for use in the simulation. First, a deserialize method is written in order to parse the serialized array produced by the PopuCreator class, and within the constructor of the class the contents of this deserialized array should be copied into a new array of Individuals- named in this case *demos*- so that the code is more succinct. Once this is done, the instance variables of the class are declared- these include the simulation parameters and the array of transmission probabilities generated by Lum et al. from the Daillaire data set [5]. The coded implementation of the beginning of the JailsbyJava class is shown below in Figure 1.2.

```
import java.io.*;
import java.util.*;
import java.lang.Math.*;

import cern.jet.*;
import cern.colt.*;

public class JailsbyJava
{
    /* Simulation Parameters */
    static int start_iter = 100;
    static int end_iter = 200;
    static double percent = .01;
    static int num_sims = 250;
    static int min_age = 15;
    static double random_prob = Math.pow((1.0-(1.0-0.01)),(1.0/50.0*12.0));
    String race = "white";
    Individual[] demos = new Individual[8856];

    static double[] famTransProbs = {0.00085, 0.00347, 0.01129,
        0.01133, 0.00801, 0.00437,
        0.03321, 0.03017, 0.00435,
        0.00078, 0.01696, 0.00634 };

    static List<Integer> alive = new ArrayList<>();
    static List<Integer> potentials = new ArrayList<>();
    static List<Integer> infected = new ArrayList<>();
    static List<Integer> counts = new ArrayList<>();

    public JailsbyJava() { System.arraycopy(deserialize(), 0, demos, 0, deserialize().length); }

    public static Individual[] deserialize() {
        Individual[] populo = new Individual[8856];
        try { FileInputStream inputFileStream = new FileInputStream("popu.ser");
        ObjectInputStream in = new ObjectInputStream(inputFileStream);
        Individual[] temp = (Individual[]) in.readObject();
        System.arraycopy(temp, 0, populo, 0, temp.length); }
    catch(ClassNotFoundException e) { e.printStackTrace(); }
    catch(FileNotFoundException e) { e.printStackTrace(); }
    catch(IOException i) {i.printStackTrace(); }
    return populo; }
```

Figure 1.2

Once the constructed population has finally been imported into the main simulation code, it is finally time to parse through it to generate the data we need. A method is created to generate sentence lengths based on the race parameter, the mean values for sentence lengths are collected from the Bureau of Justice Statistics based on convictions for drug offenses. The coded implementation is shown below in Figure 1.3.

```
public int generateSentence(String race) {
    float a = 1.2;
    if (race.equals("black"))
        float b = a / 17.0;
    if (race.equals("White"))
        float b = a / 14.0;
    double gam = cern.jet.random.Gamma.staticnextdouble(a, 1.0/b);
    int sentence = cern.jet.random.Poisson.nextInt(gam);
    return sentence; }
```

Figure 1.3

This method was written with the help of the Colt Project, a set of open source libraries for high performance scientific and technical computing by CERN. In particular, this method made use of the Gamma and Poisson classes derived from the Jet library and their respective methods to generate a Gamma and Poisson distribution from the available parameters.

The next series of method invoke the transmission probabilities generated from the Daillaire study [5] and demonstrated below in Figure 1.4 to generate binomial distributions depending on the parameters of agent sex and relationship to other agents. The coded implementation of infectious transmission between siblings is shown in Figure 1.5. The parameters this specific method receives are the sexes of both the agent being called upon and of the agent with the respective relationship, and selects the appropriate probability of transmission from the data. The method then uses this to draw samples from a Binomial distribution- specifically with the Binomial.nextInt(a,b) function derived from the Jet library- to generate a likelihood of transmission which can be applied to individuals in the population already having been constructed.

	women	**men**
mother	0.00085	0.003473
father	0.011288	0.011334
sister	0.008012	0.004367

brother	0.033205	0.030173
spouse	0.004347	0.000783
adult child	0.01696	0.006342

Figure 1.4

```
//infect sibling
public float infectSibling(String person_sex, String sibling_sex, int fac, float[] probs) {
float prob;
if (person_sex.equals('m')) && (sibling_sex.equals("m"))
    prob = probs[7];
if (person_sex.equals("m")) && (sibling_sex.equals("f"))
    prob = probs[5];
if (person_sex.equals("f")) && (sibling_sex.equals("m"))
    prob = probs[6];
if (person_sex.equals("f")) && (sibling_sex.equals("f"))
    prob = probs[4];
float infected = cern.jet.random.tDouble.Binomial.nextInt(1, (prob * fac));
return infected; }
```

Figure 1.5

The next important method is the initialization of the incarcerated population out of the 8856 individuals in the total population. This method relies on temporary arrays- to store the ID numbers of individuals who meet the desired parameters- namely, individuals whose attributes of time of death and time of birth would indicate them being alive at the start of the iteration, since those are the only individuals who have the possibility of being infected at that time. Once a collection of ID numbers of alive individuals has been collected, a new array based on those who have the potential to be "infected" is created, specifically those who are under or equal to forty-five years old, and the amount of people to be infected is based on real incarceration rates in California when mandatory drug sentencing became law – 0.01 / 1% for Blacks and 0.0015 / 0.15% for Whites. Once this list of infected ID numbers is created, the original population is iterated over to search for these ID numbers and each of these individuals' "incarcerated" attribute is changed to the sentence length depending on the race of the simulation. An individual's infected status makes them a candidate for incarceration but does not guarantee it. The final method which will be run within the main method of the simulation code class calls all of the other functions in the same class to generate the final data used for analysis. The final output of this method will be a comma separated value file, with the number of rows representing the total number of two-month periods (iterations) and the value in each row representing the number of incarcerated individuals during that two-month period. This method is run on a loop for a simulated fifty years, it keeps track both of the individuals who were incarcerated from the previous method and also runs the

familial infection methods for each individual in the population, creating more candidates for incarceration over time.

Results

Relative congruence with what has already been observed in reality demonstrates a model's success, and the Java based simulation successfully produces the disparity in incarceration rates between black and white individuals. The visual disparity illustrated by the following figures is jarring, but very explainable. Consider the following real life data from the Bureau of Justice Statistics, where one out of every 30 White men between the ages of 20 and 34 are incarcerated, and that figure jumps up to a shocking 1 out of 9 for Black males in the same age range. Looking at Figure 1.6, the simulation begins with an initial amount of 10 individuals incarcerated, but the epidemic doesn't seem to take off until halfway through the time period when it begins to grow by a vast amount. This can be explained by the fact that more and more individuals are being born and more families and network relationships are being created, which creates more opportunities for transmission. Once these opportunities, like open wounds, are initialized, more and more individuals are being incarcerated, which only increases the probability that their relatives and friends will be incarcerated as well. The minimum age of 17 indicates the minimum age one has to be to be sent to prison, which is kept constant between the two simulations but in reality can differ as prosecutors might seek inequitably convictions based on race regardless of age. Short downturns in the epidemic illustrate a high number of individuals being let out of prison due to their sentence ending, but these are short-lived as their relatives might have begun their own terms.

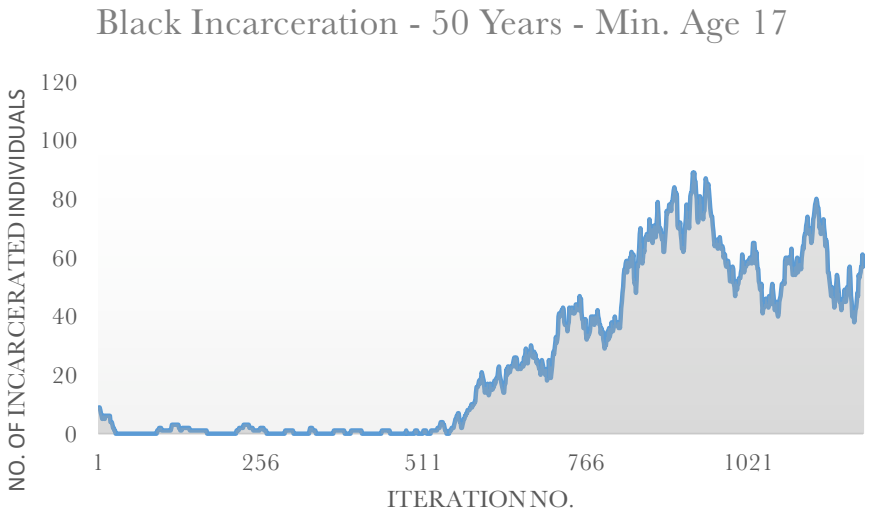

Figure 1.6

Figure 1.7 represents the white incarceration epidemic, or lack thereof. As stated previously, the parsimonious nature of the model highlights extreme disparity- which fulfills its purpose- but even then the lack of white imprisonment can be explained by the contagious nature of imprisonment that Lum et al. have explained- the fewer number of contagious individuals there are, the fewer areas of transmission there are, so there's a decreased probability of more individuals being infected and sent to prison. That is not to say that whites in the simulation are not sent to prison at all, Figure 1.7 suggests the opposite, it's merely that the initial lower opportunities of transmission and lower sentence lengths prevent severe infection from taking place, so any upward spikes in incarceration rates tend to be short-lived since white sentences tend to be more brief, and are not met with more incarceration since it is less likely the relatives or friends of these individuals will have been infected. Figure 1.10 reports that the mean quantity of incarcerated individuals in the white incarceration period is 2, but this is unfortunately skewed by the fact that there are indeed years where there are 0 white people incarcerated. This is probabilistically unlikely in real life, but in a parsimonious model where there are only a few thousand individuals alive during a certain time period within the simulation, it is not implausible in such a context.

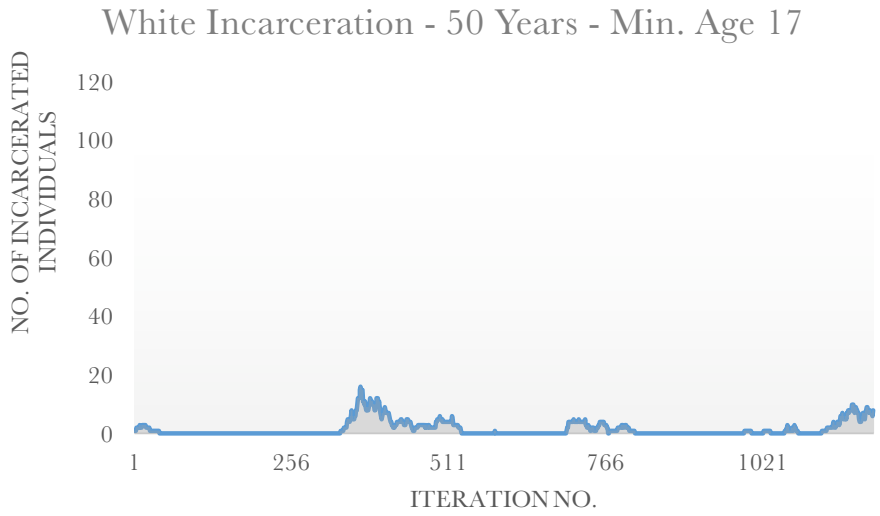

Figure 1.7

These figures demonstrate the stylized fact of incarceration, but the advantage of a computer simulation is the mutability of our parameters- thus, we can envision a possible method of closing this inequality gap- this is perhaps the most important part of this research. Let's see what happens

if we were to equalize the sentence lengths for both blacks and whites, this experiment is demonstrated in Figure 1.8.

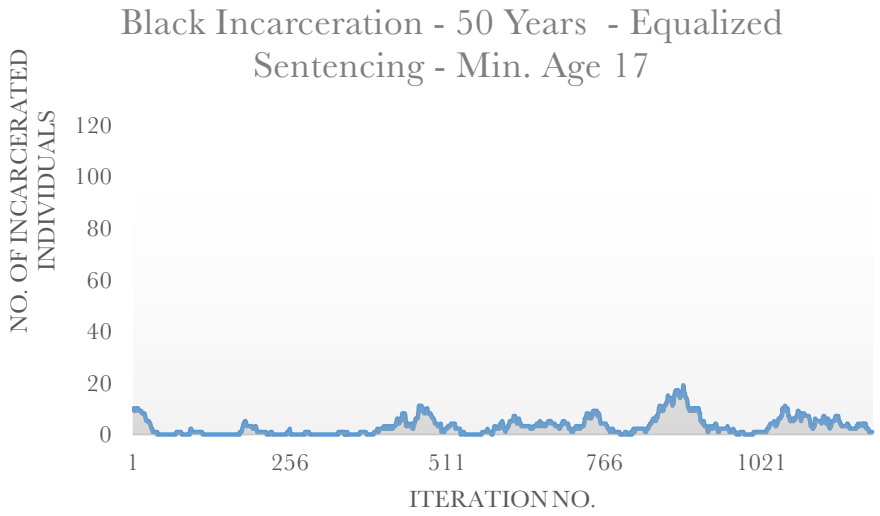

Figure 1.8

At first glance, even with a disparity still existing due to preexisting conditions, the incarceration total per iteration seems to have decreased significantly. This is the result of equalizing sentencing i.e. inputting initial parameters of 14.0 months for both races, perhaps the most idealistic initiative when it comes to wanting to close the inequality gap. This can still be explained rather simply, however- it all comes down to how many loci of transmission exist within a population at a given time, and decreased overall sentence lengths will decrease the likelihood of transmission. This can be explained on a qualitative level when it comes to the length of prison sentences in real life- the sooner the incarcerated individual returns home, the probability of a more stable family life, more sustainable household income and being exposed to criminal factors in prison decreases, it also reduces the likelihood of recidivism. Figure 1.9 explores a similar possibility, but since a lot of sentence determination is based on perceived bias- such an inherent racism would explain the inequity in punishment for the same crime- it initializes the simulation with a mean sentence length parameter of 15.5, higher than the 14.0 of the most idealistic scenario but still lower than the 17.0 of the original simulation. The results seem to demonstrate that even though incarceration tends to happen more often throughout the entirety of the time period, as opposed to the original black simulation when it rose after a certain point in time, there appear to be less individuals overall

being incarcerated. This can be explained, once again, by the decreased factors of transmission which occurs due to decreased sentence lengths.

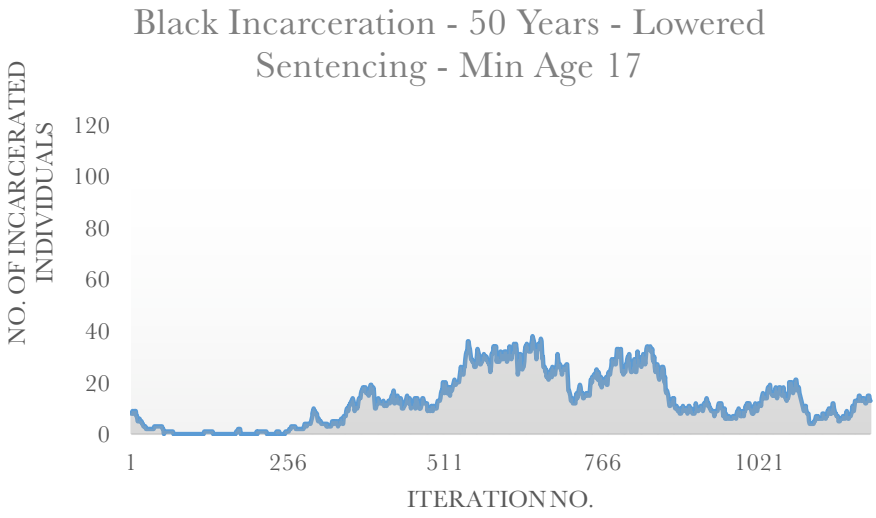

Figure 1.9

Figure 1.10 displays the mean totals of incarceration individuals per type of simulation, where 1 represents the default black incarceration simulation, 2 represents the default white incarceration simulation, 3 represents the equalized sentencing black incarceration simulation and 4 represents the lower sentencing black incarceration simulation. These results demonstrate that, even when accounting for unequal starting points, it is still worth implementing some sort of sentencing regulation or putting in place individuals less likely to impose harsher sentencing on black individuals, since it would still end up reducing the rate of transmission of incarceration by a significant amount where it would lessen the gap between white and black incarceration rates and keep more individuals out of prison. Figure 1.11 which follows immediately after explores a different scenario where the black community had the same initial incarceration rate as the white community- 0.15%- but sentencing was still at a harsh 17.0. The results are still devastating, as the total amount of incarcerated individuals shoots up near the end of the time period, which seems to further reinforced the stylized fact of the relationship between inequitable sentencing and total incarceration, no matter the initial starting point of the population.

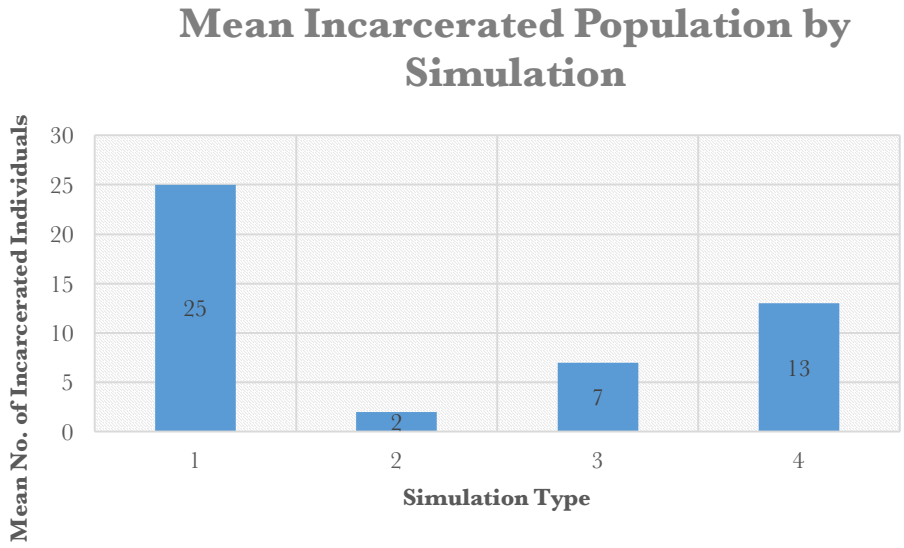

Figure 1.10

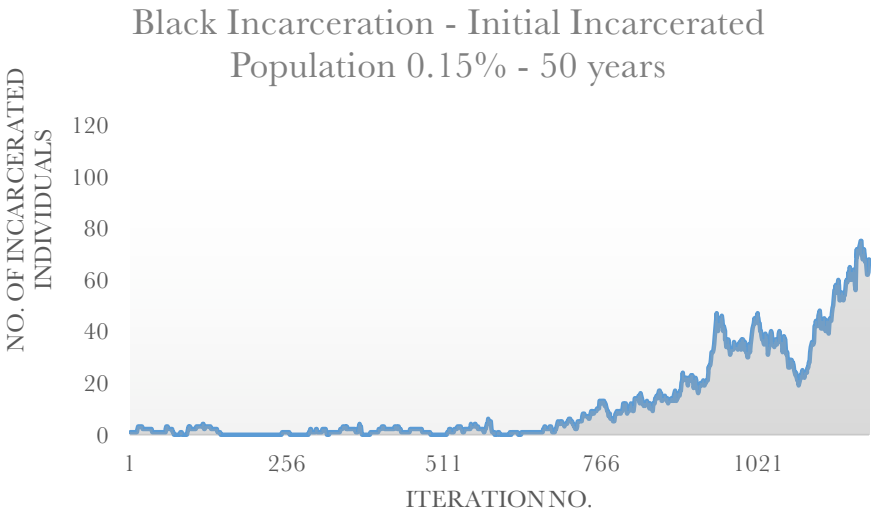

Figure 1.11

Discussion and Conclusions

One takeaway from this implementation is that it is both feasible and viable to port simulations of specific phenomena to different programming languages, provided the appropriate data structures and statistical functions are available. Creating simulations using alternative programming languages opens the door for both more efficient implementation of a simulation and a multitude of other ways to modify or display the simulation. One downside is that there is no guaranteed one-to-one correspondence in respect to the results provided by simulations, due to

differences in how even similar functions between programming languages can parse the same data. When it comes to the goal of illustrating the hypothesis that inequitable sentencing produces large disparities in incarceration rates in the long run, however, the discrepancies become irrelevant, but may come into focus as developers construct more precise models.

The creation of this model within a new language also reinforces the principles of construction of an agent-based model according to Windrum [11], one only has to create a population of individuals with clearly defined interactions and attributes and allow them to interact in order to generate aggregate data from the bottom up. Not only was this fulfilled, but the parsimonious nature of this model- the unwillingness of this model to be overwhelmingly complex and introduce superfluous variables unnecessary to simulate the observed phenomenon- has permitted the simulation of at least two methods of alleviating racial disparities in incarceration- either attempt to equalize sentencing as much as possible, or improve the initial conditions of black communities as much as possible. The specific policy implementation of such a solution, of course, ought to opt toward a two-pronged approach of both putting in place measures to deviate away from harsh sentencing while also granting pardons and reducing mandatory minimums for drug offenses, which is what this simulation has modelled in the first place. The ultimate goal should be to, as this model as pointed out, reduce the possibilities of transmitting the likelihood of being incarcerated. Figure 1.11 has demonstrated that even when blacks and whites are at the same starting point, with the same percentage of the population being incarcerated from the beginning, racial bias takes its toll and causes the epidemic to surface anyway.

Lum et al. have listed [8] many possible extensions to their model- such as simulating mixed-raced communities or adding fertility rate as a parameter since fertility rate could increase the likelihood of transmission due to the possibility of more children- but if models such as these extend beyond the realms of Java and Python and incorporate more libraries and tools, the primary focus should be to figure out ways to inoculate incarceration, to cure the prison epidemic and prevent the contagion from spreading. For years, agent-based models have helped us comprehend phenomena, but as our knowledge of stylized facts expands these models can grant us agency.

References

1. Agnew, Robert. "Foundation for a General Strain Theory of Crime and Delinquency." Criminological Theory 3.1 (1997): 1.

2. Alexander, Michelle. The new Jim Crow: Mass incarceration in the age of colorblindness. The New Press, 2012.
3. Beckett, Katherine, Kris Nyrop, and Lori Pfingst. "Race, Drugs, And Policing: Understanding Disparities In Drug Delivery Arrests*." Criminology 44.1 (2006): 105-137.
4. Birks, Daniel, Michael Townsley, and Anna Stewart. "GENERATIVE EXPLANATIONS OF CRIME: USING SIMULATION TO TEST CRIMINOLOGICAL THEORY*." Criminology 50.1 (2012): 221-254.
5. Dallaire, Danielle H. "Incarcerated mothers and fathers: A comparison of risks for children and families." *Family relations* 56.5 (2007): 440-453.
6. Epstein, Joshua M., and Robert Axtell. Growing artificial societies: social science from the bottom up. Brookings Institution Press, 1996.
7. Hommes, Cars H. "Modeling the stylized facts in finance through simple nonlinear adaptive systems." *Proceedings of the National Academy of Sciences* 99.suppl 3 (2002): 7221-7228.
8. Lum, Kristian et al. "The Contagious Nature of Imprisonment: An Agent-Based Model to Explain Racial Disparities in Incarceration Rates." Journal of the Royal Society Interface 11.98 (2014): 20140409. PMC. Web. 22 June 2015.
9. Mitchell, Ojmarrh. "A meta-analysis of race and sentencing research: Explaining the inconsistencies." Journal of Quantitative Criminology 21.4 (2005): 439-466.
10. Windrum, Paul, Giorgio Fagiolo, and Alessio Moneta. "Empirical validation of agent-based models: Alternatives and prospects." Journal of Artificial Societies and Social Simulation 10.2 (2007): 8.